Benchmarking Graph-Processing Platforms: A Vision

Yong Guo
TU Delft
The Netherlands
Yong.Guo@tudelft.nl

Ana Lucia Varbanescu
University of Amsterdam
The Netherlands
A.L.Varbanescu@uva.nl

Alexandru Iosup
TU Delft
The Netherlands
A.Iosup@tudelft.nl

Claudio Martella
VU University Amsterdam
The Netherlands
claudio.martella@vu.nl

Theodore L. Willke
Systems Architecture Lab
Intel Corporation, USA
theodore.l.willke@intel.com

ABSTRACT

Processing graphs, especially at large scale, is an increasingly useful activity in a variety of business, engineering, and scientific domains. Already, there are tens of graph-processing platforms, such as Hadoop, Giraph, GraphLab, etc., each with a different design and functionality. For graph-processing to continue to evolve, users have to find it easy to select a graph-processing platform, and developers and system integrators have to find it easy to quantify the performance and other non-functional aspects of interest. However, the state of performance analysis of graph-processing platforms is still immature: there are few studies and, for the few that exist, there are few similarities, and relatively little understanding of the impact of dataset and algorithm diversity on performance. Our vision is to develop, with the help of the performance-savvy community, a comprehensive benchmarking suite for graph-processing platforms. In this work, we take a step in this direction, by proposing a set of seven challenges, summarizing our previous work on performance evaluation of distributed graph-processing platforms, and introducing our on-going work within the SPEC Research Group's Cloud Working Group.

Categories and Subject Descriptors

C.4 [**PERFORMANCE OF SYSTEMS**]: Measurement techniques

Keywords

Graph processing; Benchmarking; Performance; Experimentation

1. INTRODUCTION

Graph processing is of increasing interest for many revenue-generating applications and scientific areas, such as social networking, bioinformatics, and online retail and online gaming. To answer to the growing diversity of graph datasets and graph-processing algorithms, developers and system integrators have created a large variety of graph-processing *platforms*—which we define as the combined hardware, software, and programming system that is being used to complete a graph processing task [17]. Although these platforms are already much used, it is currently difficult to decide on deploying a new platform and to tune existing deployments, due to a lack of comprehensive understanding of the performance of these platforms. To gain more and more in-depth knowledge about graph-processing platforms, and to enable their comparison, we envision a comprehensive benchmarking suite, which is the focus of this work.

At least three dimensions of diversity complicate gaining knowledge about the performance of graph-processing platforms [17]: dataset, algorithm, and platform diversity. *Dataset diversity* derives from the data deluge we are experiencing—graphs from hundreds of areas, from genomics to consumer profiles, from social gaming networks to business decision support, with periodic updates and different data structures. *Algorithm diversity* is a consequence of the many different goals of processing graphs, with a variety of graph algorithms for calculating basic graph metrics [22], for traversing graphs [24, 27, 30], and for predicting graph evolution [20], etc. *Platform diversity* is the result of the uncoordinated effort of a multitude of developers, to answer their community of users, sometimes additionally influenced by the wide diversity of infrastructure (compute and storage systems); this has led, for example, to graph processing with generic platforms such as Hadoop [29] and YARN [5], with distributed graph-processing platforms such as Giraph [2] and GraphLab [21], etc.

Not understanding the performance of the graph-processing platform can lead to significant time- and revenue-loss, and may eventually even limit the growth, for the entire community. For example, it took many tries, but by now the community agrees that using Hadoop as a graph-processing platform generally leads to poor performance [17, 23, 25]. For specific datasets and algorithms, platforms may experience crashes, due to inefficient data structures [17, 25], network-stack overloads [13, 17], etc. For different datasets and algorithms, platforms may exhibit very different relative execution-profiles [17]; tuning for one input workload may be lead to sub-optimal results for another.

Although sorely needed, relatively few studies focus on the performance of graph-processing platforms; the few that do use few datasets, few algorithms, and few metrics to characterize the platform. The de-facto state-of-the-art in benchmarking, Graph500 [15] and its energy-aware relative Green Graph500, is based on a single dataset type, on a single algorithm, and on just a few metrics. Most empirical comparisons of graph-processing platforms occur in articles focusing on new designs, and thus may lack depth and objectivity. For example, a recent study of Trinity [25] focuses on novel techniques for in-memory graph processing, but the experiments, although commendable, only compares Trinity with Giraph and PBGL [16], only uses three algorithms, and only reports average response time and memory usage.

We formulate in this work our vision for benchmarking graph-processing platforms that considers *all three* sources of diversity. Our work consists of defining a comprehensive evaluation process, of selecting important performance metrics, representative datasets, and typical algorithms, of conducting and executing the experiments, and of analyzing and reporting the results. This vision extends our own recent work [17] and other recent experience reports [6, 11, 12] with a set of methodological and practical challenges, a survey of graph-processing use, and a call to combine efforts within the SPEC Research Group's Cloud Working Group. It also extends traditional benchmarking with graph-specific elements and with aspects related to new infrastructures (e.g., clouds). Our main contribution is three-fold:

1. We discuss seven methodological and practical challenges in benchmarking graph-processing platforms (Section 2).
2. We summarize our work towards evaluating and benchmarking graph-processing platforms, including understanding the workloads of graph processing, and our previous work [17] about proposing a method for benchmarking graph-processing platforms and a first comprehensive performance comparison of six popular graph-processing platforms (Section 3).
3. We call for the entire community to participate in the creation of a benchmark and give examples of on-going work (Section 4).

2. OUR VISION FOR BENCHMARKING GRAPH-PROCESSING PLATFORMS

Benchmarking is a traditional approach to evaluate the performance of systems, with many well-known challenges: simplicity, cost- and time-effectiveness, verifiability, etc. However, benchmarking systems under different application can lead to specific challenges. In this section, we discuss seven challenges in benchmarking graph-processing platforms. Although there are more challenges to resolve, we argue that these would lead to a good benchmarking process, similar to what has been achieved by the TPC and SPEC communities for benchmarking databases, CPU power and energy, etc.

2.1 Methodological Challenges

Challenge 1. Evaluation process: Traditionally, it is a challenge to define an evaluation process that would define an equivalent benchmarking process for each platform (for example, not controlling the amount of tuning can lead to a war-of-wizards). For graph-processing platforms, the evaluation process needs to fairly define at least the data format, realistic processing workflows, and the multi-tenancy rules—although these concepts have been considered in the past, they need revisiting for graph processing. Although the mathematical notion of a graph allows for only a few varieties, in graph-processing applications we have seen various data structures, input formats, and number of dimensions for the dataset. Similarly to the idea that a single query may expand in several data operations, in graph-processing it is likely that processing workflows comprised of several atomic operations (single algorithms) is representative of the typical analysis task; the evaluation process should also include such workflows. Because graph-processing platforms are typically serving multiple users, much like modern databases and distributed batch-processing systems, the evaluation process should also consider how the workloads of multiple system tenants influence each other.

Challenge 2. Selection and design of performance metrics: To serve more users, one important issue for benchmarking graph-processing platforms is to provide performance metrics for a variety of platform characteristics. Typical performance metrics such as execution time, resource utilization, scalability, system overhead, power consumption, cost, etc., may be included. To compare platforms on top of various types and amounts of hardware resources (e.g. number of cores or size of memory), new normalized metrics may need to be defined and adopted. For example, Graph500 introduces the graph-specific metric *traversed edges per second* (TEPS). We argue that there is much room for metric definition. As another example, as the field spans database, parallel, and distributed systems, normalized metrics for weak and strong scaling of possibly heterogeneous platforms need to be devised. Moreover, there is a need to adapt traditional metrics to an elastic infrastructure, for example because the local infrastructure may be complemented with nodes leased temporarily from Infrastructure-as-a-Service clouds [14].

Challenge 3. Dataset selection: Selecting a representative dataset is a traditional problem in benchmarking, which requires revisiting for each new domain. As we present in section 1, graphs may differ significantly in size, structure, directivity, connectivity, etc. The main goal of the dataset selection is to choose relevant graphs with different characteristics; to make the benchmark time- and cost-effective, they should also be *few*, easy to generate at different scales (see Challenge 5), and stored in a similar format (see Challenge 1). Additionally, this challenge also requires that the selected graphs should be able to stress bottlenecks of graph-processing platforms.

Challenge 4. Algorithm coverage: Similar to Challenge 3, we find challenging the selection of a representative, reduced set of graph-processing algorithms, which may stress diverse components of the graph-processing platform. To reduce the number of algorithms, it should be possible to divide them into classes, based on their functionality and to select representative algorithms from each class. This solution also has some limitations: how to define the classes? how to select a representative algorithm from a class? how to allow future algorithms to participate in the benchmark? etc.

2.2 Practical Challenges

Challenge 5. Scalability of evaluation and selection processes: It is challenging to allow the users of a benchmark—developers and integrators of platforms, graph analysts, etc.—to cope with the scale of either the evaluation or the selection processes.

For the evaluation process, we aim at benchmarking platforms deployed on both large-scale infrastructure (e.g., wide-area multicluster systems, large data centers, supercomputers, etc.) and small-business infrastructure (e.g., clusters of only a few nodes, a single albeit powerful machine). Thus, the benchmarking suite, and in particular the input datasets, should match various operational scales—for datasets, from megabytes to petabytes. Currently, few real-world graphs of petabyte scale are available for bechmarking activities, and even datasets of hundreds of gigabytes are rare. Graph generators could produce graphs for testing of the required scale, for example the Kronecker generator used in Graph500, but pose important parallel/distributed computing challenges and may not have the characteristics of real-world graphs.

For the selection process, a community-oriented benchmarking process should also be able to match the possibly hundreds of metrics with the interests of the users. For graph analysts, a specific application may need to be matched against an entire database of benchmarking results, and a few most-promising systems should be selected. For system integrators, it would be helpful to identify which algorithms and graphs can stress the system for each selected metric. We believe that designing a community database of open results would be beneficial in addressing this challenge, but its design should be able to accommodate a wide variety of settings and thus remains an open challenge.

Challenge 6. Portability: As we discussed in the methodological challenges, the benchmarking suite includes a number of performance metrics, algorithms, and graphs. When benchmarking a platform, the graph-processing algorithms need to be re-implemented based on the platform's programming language and model and, possibly, also based on infrastructure characteristics. Re-implementing algorithms correctly and re-configuring reasonably of a platform need a solid experience of programming and a detailed understanding of the platform. The challenge is, thus, to design a benchmarking suite that balances the portability requirements with all the desired features.

Challenge 7. Result reporting: Another non-trivial practical aspect is to report benchmarking results, which should be done according to a precisely defined format. Comprehensive and standardized reports traditionally facilitate the understanding and the comparison of the performance of platforms. When users consider several performance metrics when comparing graph-processing platforms, a mechanism to combine the results from different performance metrics and report a single result may not be straightforward—in our experience [17], none of the distributed graph-processing platforms can deliver the best performance across all datasets and algorithms, even for the same metric. Other communities have faced this challenge in the past and were able to solve it. For example, SPEC benchmark results can include a full disclosure of the system configuration parameters; SPEC users can report both baseline (not tuned) and peak (tuned) performance results of systems. However, it took years of development and effort to achieve this by SPEC benchmarks.

3. TOWARDS ACHIEVING OUR VISION

In this section we summarize our work on benchmarking and evaluating graph-processing platforms. We emphasize on two aspects: understanding graph-processing requirements, and proposing a method for benchmarking and performance evaluation of graph-processing platforms.

3.1 Understanding Graph-Processing Workloads

We believe that a variety of workloads—in this work, defined as combinations of datasets and algorithms, inter-dependencies between inputs of an algorithm and outputs of another (effectively, a workflow structure), and a general process for submission for execution to the platform—have appeared in the context of graph processing, with variety due to application domain, independent communities developing their own workload policies, and perhaps even due to the availability of different processing platforms. However, there is no common repository of algorithms, datasets, workflows, etc., which makes Challenges 1–4 difficult to address.

To address Challenges 2–4, we have conducted comprehensive literature surveys of metrics, datasets, and algorithms used in practice (not reported in our previous work [17]). We have specifically targeted articles about graph processing published in top research conferences in the fields of databases, distributed systems, and information retrieval, such as SIGMOD, (P)VLDB, HPDC, etc.; our assumption is that these prestigious conferences have attracted a knowledgeable audience, and represent a meaningful sample of industry and scientific efforts in graph processing. We have searched for articles including the words "graph processing", "social network", etc. In total, we have extracted information from 124 articles published in 10 representative conferences over the period 2009–2013.

Table 1: Survey of graph algorithms.

Class	Typical algorithms	Number	Percentage [%]
General Statistics	Triangulation [28], Diameter [19], BC [26]	24	16.1
Graph Traversal	BFS, DFS, Shortest Path Search	69	46.3
Connected Components	MIS [8], BiCC [10], Reachability [9]	20	13.4
Community Detection	Clustering, Nearest Neighbor Search	8	5.4
Graph Evolution	Forest Fire Model [20], Preferential Attachment Model [7]	6	4.0
Other	Sampling, Partitioning	22	14.8
Total		149	100

Challenge 2: Few performance metrics are tested and reported in these articles; most of the performance evaluation focuses only on the job execution time, and, seldomly, they report on metrics such as scalability and throughput, and on memory consumption.

Challenge 3: We have observed that a large number of datasets, from various areas, are processed in previous research. In general, these graphs can be divided into two categories, real-world and synthetic. Real-world graphs are collected by researchers from their own applications or from public graph archives, for example, from the Stanford Network Analysis Project (SNAP) [4]) and from the Game Trace Archive (GTA) [18]. Synthetic graphs with different structures are generated from several graph generators, such as Kronecker, Erdős-Rényi, and R-MAT. Notably, the maximum size reported in these articles for real graphs (1.7 billion vertices and 7.9 billion edges) is significantly *smaller* than that of synthetic graphs (274.9 billion vertices and 4.4 trillion edges).

Challenge 4: We have found that a large variety of graph-processing algorithms are reported in practice. Table 1 summarizes the algorithms identified in our survey—149 in 124 articles. We categorize these algorithms into several groups by functionality, consumption of resources, etc. Almost half of the articles we survey (46.3%) use some form of graph traversal in their experimental work. The next most-represented classes of algorithms compute general graph statistics (16.1%), and extract or use connected components (13.4%). A variety of algorithms are present in less than 3% of the articles we have surveyed; together, they account for over a seventh (14.8%) of the articles.

3.2 Method for and Performance Evaluation of Graph-Processing Platforms

In our previous work [17], we have taken first steps to defining an empirical method for benchmarking graph-processing platforms (Challenge 1), applied it in practice by porting the benchmarks to six different platforms (Challenge 6), and reported on the experience (Challenge 7). We summarize here our main achievements, towards addressing the challenges.

Challenge 1: We have used both synthetic (Kronecker) and real datasets (SNAP and GTA): both directed and undirected graphs, with unweighted edges, and vertices represented by numeric identifiers. We have employed a vertex-based data storage; only atomic operations (single algorithms, so no workflows); and a single user (no multi-tenancy). We intend to extend our experiments towards workflows and multi-tenancy. From our survey (see Section 3.1), we have selected one exemplary algorithm for each of the most-represented five algorithmic classes.

Challenge 6: We have ported the five graph-processing algorithms on six popular graph-processing platforms: Hadoop, YARN, Stratosphere, Giraph, GraphLab, and Neo4j [3]. The first five platforms are distributed systems. We select the single-node platform Neo4j as a reference for comparison. We have also analyzed the time taken to port the codes to each platform; in total, between days and weeks. We have further deployed all these platforms in a cluster of up to 50 computing nodes from the DAS4 cluster. [1].

Challenge 7: We have reported metrics on four typical performance aspects: *raw processing power*, defined as the ability of a platform to process (large-scale) graphs; *resource utilization*, defined as the ability of a platform to efficiently utilize the resources it has; *scalability*, defined as the ability of a platform to maintain its performance behavior when resources are added to its infrastructure; and *overhead*, defined as the part of wall-clock time the platform does not spend on true data processing. The metrics include traditional system parameters (e.g., job execution time, the CPU and network load, and the OS memory consumption); normalized graph-specific metrics, such as TEPS; etc.

4. A CALL TO ARMS

Graph processing is rapidly expanding in volume and diversity of datasets, algorithms, and overall usage. To prevent that the domain becomes too fragmented, and to allow graph analysts and system integrators to compare existing platforms, we envision the creation of a benchmarking suite for graph-processing platforms.

We have identified seven main challenges, and conducted work in understanding graph-processing workloads and in the comprehensive performance evaluation of six popular platforms, but much remains to be done. We urge the community to join forces and conduct peer work in the SPEC Research Group's Cloud Working Group. The SPEC Research Group (RG) is a new group within the Standard Performance Evaluation Corporation (SPEC). The Cloud Working Group[1] (CWG) is a branch of the SPEC RG that aims to develop the methodological aspects of cloud benchmarking—among its activities, we have included Graph Processing as a Service, which relies on quantifiable service performance and thus benchmarking. Within the Cloud Working Group, we are currently addressing all the challenges introduced in Section 2, but in particular:

1. *Challenge 1:* Defining workloads that include processing workflows and multi-tenancy aspects.
2. *Challenges 2–4:* Through a survey of relevant graph analysts and system integrators[2], understanding the metrics, datasets, and algorithms used in practice.
3. *Challenge 1, 2, and 7:* Evaluating and reporting on the platform–storage engine relationship.

5. REFERENCES

[1] DAS4. `http://www.cs.vu.nl/das4/`.

[2] Giraph. `http://giraph.apache.org/`.

[3] Neo4j. `http://www.neo4j.org/`.

[4] SNAP. `http://snap.stanford.edu/index.html`.

[5] YARN. `http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html`.

[6] D. D. Abreu, A. Flores, G. Palma, V. Pestana, J. Piñero, J. Queipo, J. Sánchez, and M.-E. Vidal. Choosing between graph databases and rdf engines for consuming and mining linked data. In *COLD*, 2013.

[7] A.-L. Barabási and R. Albert. Emergence of scaling in random networks. 1999.

[8] A. Buluç, E. Duriakova, A. Fox, J. R. Gilbert, S. Kamil, A. Lugowski, L. Oliker, and S. Williams. High-Productivity and High-Performance Analysis of Filtered Semantic Graphs. In *IPDPS*, 2013.

[9] J. Cai and C. K. Poon. Path-hop: efficiently indexing large graphs for reachability queries. In *CIKM*, 2010.

[10] G. Cong and K. Makarychev. Optimizing Large-scale Graph Analysis on Multithreaded, Multicore Platforms. In *IPDPS*, 2012.

[11] M. Dayarathna and T. Suzumura. Xgdbench: A benchmarking platform for graph stores in exascale clouds. In *CloudCom*, pages 363–370, 2012.

[12] B. Elser and A. Montresor. An evaluation study of bigdata frameworks for graph processing. In *IEEE BigData*, 2013.

[13] S. Ewen, K. Tzoumas, M. Kaufmann, and V. Markl. Spinning fast iterative data flows. *PVLDB*, 5(11):1268–1279, 2012.

[14] B. Ghit, N. Yigitbasi, and D. Epema. Resource Management for Dynamic MapReduce Clusters in Multicluster Systems. In *SC|12 MTAGS*, 2012. Best paper award.

[15] Graph500. `http://www.graph500.org/`.

[16] D. Gregor and A. Lumsdaine. The Parallel BGL: A Generic Library for Distributed Graph Computations. *POOSC*, 2005.

[17] Y. Guo, M. Biczak, A. L. Varbanescu, A. Iosup, C. Martella, and T. L. Willke. How well do graph-processing platforms perform? an empirical performance evaluation and analysis. In *IPDPS*, 2013. `http://www.pds.ewi.tudelft.nl/~iosup/perf-eval-graph-proc14ipdps.pdf`.

[18] Y. Guo and A. Iosup. The Game Trace Archive. In *NetGames*, 2012.

[19] W. Jiang and G. Agrawal. Ex-MATE: Data Intensive Computing with Large Reduction Objects and Its Application to Graph Mining. In *CCGRID*, 2011.

[20] J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over Time: Densification Laws, Shrinking Diameters and Possible Explanations. In *SIGKDD*, 2005.

[21] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, and J. M. Hellerstein. Distributed GraphLab: A Framework for Machine Learning and Data Mining in the Cloud. In *VLDB*, pages 716–727, 2012.

[22] A. Lugowski, D. M. Alber, A. Buluç, J. R. Gilbert, S. Reinhardt, Y. Teng, and A. Waranis. A Flexible Open-Source Toolbox for Scalable Complex Graph Analysis. In *SDM*, 2012.

[23] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. Pregel: A System for Large-scale Graph Processing. In *SIGMOD*, pages 135–146, 2010.

[24] D. Merrill, M. Garland, and A. S. Grimshaw. Scalable GPU graph traversal. In *PPOPP*, 2012.

[25] B. Shao, H. Wang, and Y. Li. Trinity: A distributed graph engine on a memory cloud. In *SIGMOD*, 2013.

[26] J. Shun and G. E. Blelloch. Ligra: a lightweight graph processing framework for shared memory. In *PPOPP*, 2013.

[27] E. Solomonik, A. Buluç, and J. Demmel. Minimizing Communication in All-Pairs Shortest Paths. In *IPDPS*, 2013.

[28] N. Wang, J. Zhang, K.-L. Tan, and A. K. H. Tung. On Triangulation-based Dense Neighborhood Graphs Discovery. *VLDB*, 2010.

[29] T. White. *Hadoop: The definitive guide*. O'Reilly Media, Inc., 2012.

[30] B. Wu and Y. Du. Cloud-Based Connected Component Algorithm. In *ICAICI*, pages 122–126, 2010.

[1]http://research.spec.org/working-groups/rg-cloud-working-group.html

[2]You are invited to participate, `http://goo.gl/TJwkTg`.